

Créez des applications Delphi compatibles Windows Vista - Mise à jour

par Nathaniel Woolls ([Victory Technologies](#)) Traduction par Olivier Lance ([Accueil](#))

Date de publication : 24 mars 2007

Dernière mise à jour :



Windows Vista est maintenant disponible depuis quelques mois. Si vous avez su tirer partie du premier article à ce sujet, cette seconde traduction vous intéressera sans doute.

Nathaniel Woolls a complété son  **article original** en réaction aux retours qu'il en a reçus, et pour corriger/améliorer certains codes. C'est la traduction de cette  **seconde version** de *Creating Windows Vista Ready Applications* que je vous propose ici, afin que vous puissiez profiter du travail qui a été effectué ces derniers mois par les delphistes anglophones les plus actifs.

Un grand merci à Nathaniel Woolls pour m'avoir donné l'autorisation à deux reprises de traduire et de vous proposer librement la version française de ses articles.

- I - Réintroduction
- II - Réparation de polices
- III - Ajout pour les polices
- IV - TreeView de style Vista
- V - L'astuce de PopupParent
- VI - Activation étrange
- VII - Enumérer des fenêtres
- VIII - Les raccourcis avec ALT cachent des composants
- IX - Un Dialogue de Tâche multi-plateforme
- X - Conclusion et problèmes en suspens

I - Réintroduction

J'ai été pour le moins étonné par l'accueil qui a été réservé à mon **article original** ( **version traduite**) sur la création d'applications compatibles **Windows Vista** avec Delphi. J'ai été très flatté d'être inclus dans la *Delphi Hour* (enregistrement disponible  [ici](#)).

Le but de cet article est de fournir des informations supplémentaires et mises à jour quant à la création d'applications Delphi compatible Vista. Certains codes de l'article original présentent quelques problèmes que nous examinerons et réglerons, nous explorerons de nouveaux codes pour utiliser les polices, des codes nous permettant de tirer parti du nouveau style de TreeView de Vista, et plus encore.

Cet article suppose que vous avez lu le **premier article** ; les exemples de code se basent sur ceux développés dans ce dernier.

II - Réparation de polices

Mes remerciements à David Rose qui a aidé à trouver un problème dans **SetVistaFont()**. J'avais initialement l'impression que modifier **Font.Name** pour une police qui n'était pas installée laisserait **Font.Name** inchangée. Ce n'est pas le cas. Notre **code original** vérifiait **Font.Name** pour voir si **Segoe UI** était correctement appliquée à la police de la fiche, ce qui ne fonctionnait pas. Notre nouveau code teste **Screen.Fonts** pour vérifier que Segoe UI est une police installée.

SetVistaFonts()

```
procedure SetVistaFonts(const AForm: TCustomForm);
begin
  if (IsWindowsVista or not CheckOSVerForFonts)
    and not SameText(AForm.Font.Name, VistaFont)
    and (Screen.Fonts.IndexOf(VistaFont) >= 0) then
  begin
    AForm.Font.Size := AForm.Font.Size + 1;
    AForm.Font.Name := VistaFont;
  end;
end;
```

Le nouveau code introduit aussi la nouvelle variable globale **CheckOSVerForFonts**, qui détermine si les modifications de la police doivent prendre effet sous Vista uniquement, ou s'ils doivent aussi s'appliquer sous les autres OS. Le code original n'aurait posé de problèmes que dans le cas où Segoe UI n'aurait pas été installée sous Vista (peu probable), mais ce code est bien plus complet et correct.

III - Ajout pour les polices

Sur le [CodeGear Developer Network](#), Daniel England a soulevé **qu'il n'aimait pas que les noms des polices soient codés "en dur"**. Bien que j'approuve, mon intention était d'essayer de conserver la plupart de ces changements au fil du code, avec l'expérience "prêt à l'emploi" que l'on retrouve avec Delphi. Delphi, par défaut, n'ajuste pas la police de la fiche afin qu'elle corresponde à la police courante de Windows. S'il le faisait, nous n'aurions pas du tout à changer nos polices vers Segoe UI sous Vista.

Ceci étant dit, le premier ajout au code concernant les polices est la procédure **SetDesktopIconFonts()**. Cette procédure peut être appelée avec un **TFont**, et l'objet de police verra ses propriétés accordées aux paramètres courants de police des icônes du bureau Windows. Dans l'exemple mis à jour, cette méthode est employée plutôt que **SetVistaFont()**.

SetDesktopIconFonts()

```
procedure SetDefaultFonts(const AFont: TFont);
begin
  AFont.Handle := GetStockObject(DEFAULT_GUI_FONT);
end;

procedure SetDesktopIconFonts(const AFont: TFont);
var
  LogFont: TLogFont;
begin
  if SystemParametersInfo(SPI_GETICONTITLELOGFONT, SizeOf(LogFont),
    @LogFont, 0) then
    AFont.Handle := CreateFontIndirect(LogFont)
  else
    SetDefaultFonts(AFont);
end;
```

Ce code tente de récupérer les paramètres de polices des icônes du bureau Windows actuel et, si cela échoue, se rabat sur les paramètres de police par défaut de Windows.

Notre second ajout au code concernant les polices est **SetVistaContentFonts()**. Cette procédure réalise la même opération que **SetVistaFonts()** mais seulement pour le contenu de composants tel que les **TMemo**, **TRichEdit** et autres du genre. Dans ce cas, la nouvelle police est, *de facto*, **Calibri**.

SetVistaContentFonts()

```
procedure SetVistaContentFonts(const AFont: TFont);
begin
  if (IsWindowsVista or not CheckOSVerForFonts)
    and not SameText(AFont.Name, VistaContentFont)
    and (Screen.Fonts.IndexOf(VistaContentFont) >= 0) then
  begin
    AFont.Size := AFont.Size + 2;
    AFont.Name := VistaContentFont;
  end;
end;
```

Un appel à cette nouvelle méthode avec un **TFont** pour paramètre impliquera le changement du nom de la police vers Calibri et l'augmentation de la taille de la police si l'application est exécutée sous Vista (ou si la variable globale **CheckOSVerForFonts** est égale à **false**).

IV - TreeView de style Vista

Le TreeView est un autre composant de l'interface graphique amélioré dans Windows Vista, et qui n'a pas été mentionné dans l'article original. Sous Windows Vista, les sélections dans le TreeView sont sous forme d'un dégradé bleu ciel plutôt que le traditionnel rectangle de sélection, et les boutons du TreeView sont de petits triangles plutôt que des signes "+".

Faire adopter ce nouveau style à nos objets **TTreeView** est relativement trivial, et le code se trouve dans l'exemple mis à jour dans la méthode **SetVistaTreeView()**.

SetVistaTreeView()

```
procedure SetVistaTreeView(const AHandle: THandle);  
begin  
  if IsWindowsVista then  
    SetWindowTheme(AHandle, 'explorer', nil);  
end;
```

Appeler cette méthode avec le handle du TreeView donnera à ce dernier l'aspect du nouveau TreeView de Vista.

V - L'astuce de PopupParent

Notre article original **introduisait un code** pour passer outre la fenêtre d'application cachée de Delphi, permettant à plusieurs nouveaux éléments de l'interface utilisateur de Vista de fonctionner correctement, comme **Flip 3D** et les aperçus en temps réel. Cependant, ces modifications nous obligent à changer la propriété **PopupParent** avant chaque appel à ShowModal pour assurer un bon agencement en profondeur de nos fiches.

En correspondant avec Eric Fortier, j'ai poussé plus loin mes recherches et tenté de voir s'il était possible de supprimer cette nécessité. Bien que ce soit possible, cela requiert des modifications du code de la VCL (similaires à nos changements de Dialogs.pas dans l'article original).

- Premièrement, faire une copie de Forms.pas et placer cette copie soit dans le répertoire des sources de votre application ou quelque part dans vos chemins de recherche
- Deuxièmement, dans cette copie de Forms.pas, allez à la ligne 4032
- Cherchez l'instruction **case** sur *LPopupMode*
- Commentez la première ligne suivant l'instruction : *//pmNone: Application.Handle;*
- Editez la seconde ligne, qui se déclenche pour la valeur pmAuto, afin qu'elle prenne également pmNone en compte : *pmNone, pmAuto:*
- Sauvegardez vos modifications sur votre copie de Forms.pas

Cela devrait vous laisser avec une instruction case autour de la ligne 4032 qui inspecte **LPopupMode** et qui, maintenant, exécute la même action à la fois pour **pmNone** et **pmAuto** plutôt que d'utiliser **Application.Handle** comme parent lorsque **PopupMode** vaut **pmNone**. C'est justement ce que l'on veut. Maintenant, la VCL tentera de trouver notre fiche parente en détectant la fenêtre active avec un minimum de logique, plutôt que de simplement utiliser **Application.Handle** (ce qui ne fonctionne pas à cause de nos modifications du premier article) lorsque **PopupMode** est égal à **pmNone**.

VI - Activation étrange

Un des effets de bord **du code présenté** dans l'article original pour régler les problèmes d'animations et d'aperçus de fenêtres est qu'une fiche en-dessous d'une autre fiche modale peut devenir visuellement active si son bouton de barre des tâches est cliqué. Notez qu'il s'agit seulement d'une altération visuelle, vous ne pouvez pas réellement interagir avec un quelconque élément de la fiche. Max Pyatnitsky a soulevé ce problème sur les **newsgroups CodeGear**, et j'ai pensé partager une version légèrement modifiée de sa solution (elle est maintenant incluse dans l'exemple mis à jour et l'exécutable compilé):

```
procedure TMainForm.WMActivate(var Message: TWMActivate);
begin
  if (Message.Active = WA_ACTIVE) and not IsWindowEnabled(Handle) then
  begin
    SetActiveWindow(Application.Handle);
    Message.Result := 0;
  end else
    inherited;
end;
```

Merci à Max Pyatnitsky pour sa solution à ce problème. Ce code active la fiche cachée **Application** quand notre fiche est activée, assurant ainsi que toute fenêtre modale prenne le focus correctement.

VII - Enumérer des fenêtres

Il y a un autre avertissement pour les utilisateurs qui auraient implémenté **le code** pour contourner la fiche cachée **Application** de Delphi. Selon Tom Nagy sur les **newsgroups CodeGear**, les développeurs qui énumèrent les fenêtres en avant-plan avec **EnumWindows()** sont susceptibles d'utiliser un code tel que celui-ci comme code de la Callback **EnumWindowsProc** :

EnumWindowsProc classique

```
if IsWindowVisible(Wnd) and  
  
  ((GetWindowLong(Wnd, GWL_HWNDPARENT) = 0) or  
   (HWND(GetWindowLong(Wnd, GWL_HWNDPARENT)) = GetDesktopWindow)) and  
  ((GetWindowLong(Wnd, GWL_EXSTYLE) and WS_EX_TOOLWINDOW) = 0) then  
begin  
  ...  
end;
```

Selon Tom il s'agirait d'un morceau de code utilisé très couramment lors de l'énumération de fenêtres. Cependant, nos modifications du flag **CreateParams** de notre fiche causera l'échec de la vérification de **WX_EX_TOOLWINDOW**. Tom propose la modification de code suivante (retirer la vérification de **WX_EX_TOOLWINDOW**) pour régler le problème :

Solution de Tom Nagy

```
if IsWindowVisible(Wnd) and  
  
  ((GetWindowLong(Wnd, GWL_HWNDPARENT) = 0) or  
   (HWND(GetWindowLong(Wnd, GWL_HWNDPARENT)) = GetDesktopWindow)) then  
begin  
  ...  
end;
```

Merci à Tom Nagy pour avoir trouvé ceci et l'avoir indiqué sur les newsgroups. Espérons que cela en aidera d'autres à éviter un bug qui pourrait s'avérer compliqué à détecter !

VIII - Les raccourcis avec ALT cachent des composants

Il existe un autre bug présent dans les applications Delphi lorsqu'elles tournent sous Vista. Ce bug a été **rapporté** sur **Quality Central** et possède plusieurs solutions. Le problème est que, lorsque la touche ALT est appuyée, certains **TWinControl** peuvent disparaître de votre fenêtre jusqu'à ce que leur rafraichissement soit forcé. Il existe **ici** un composant qui règle ce problème. J'ai essayé les modifications de code dans Controls.pas **présentées ici** et pour ma part elles n'ont pas réglé le problème. Ceci dit le composant ci-dessus semble fonctionner correctement.

IX - Un Dialogue de Tâche multi-plateforme

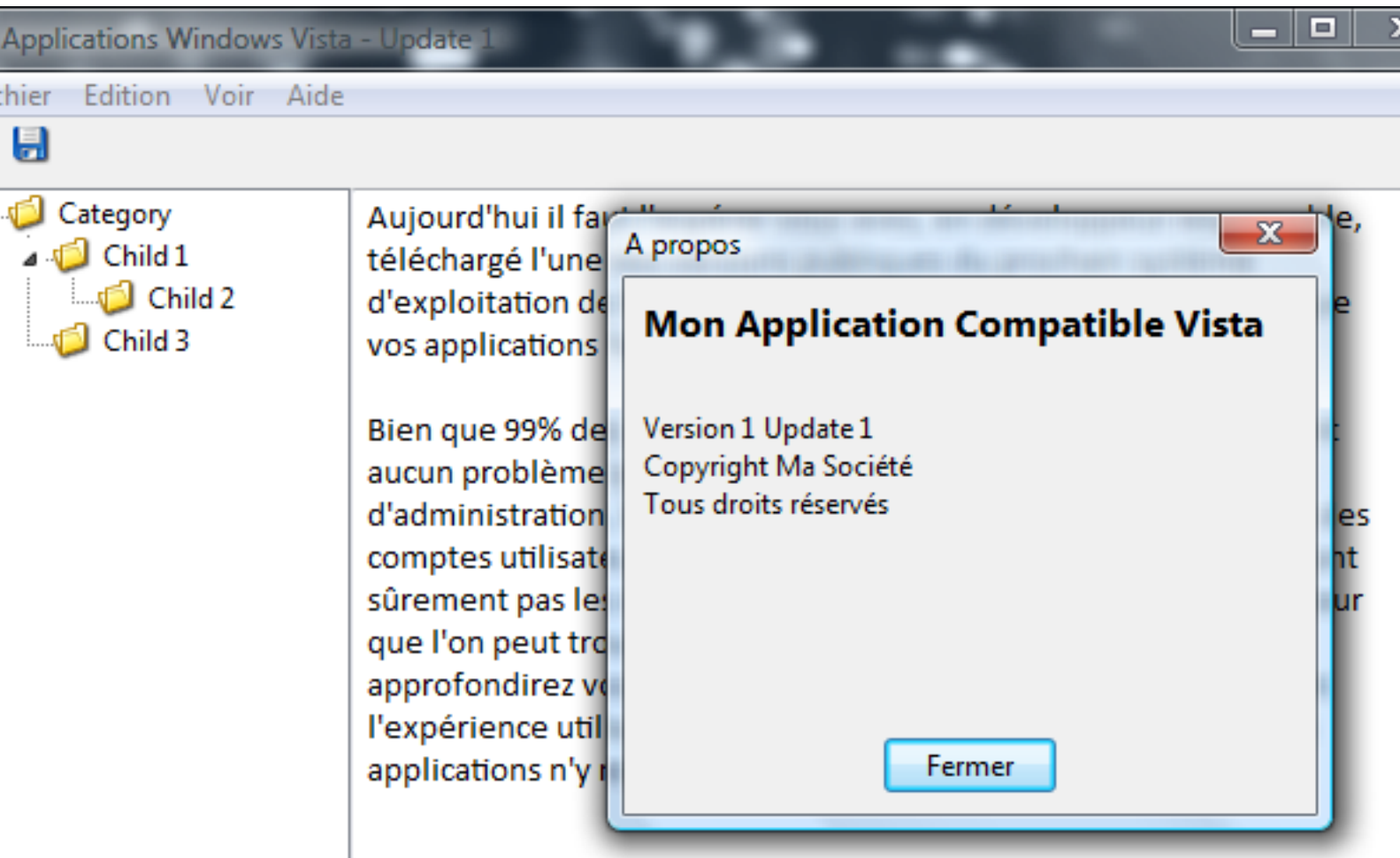
L'article original introduisait le concept des **Dialogues de Tâche** dans Vista, et comment utiliser ces nouveaux styles de dialogues depuis vos applications Delphi. Bien que ces nouveaux Dialogues de Tâche soient un très bon ajout à Windows Vista, il n'est pas aisé de supporter à la fois les fonctionnalités des nouveaux Dialogues de Tâche et de revenir à d'autres dialogues supportés sous XP et plus récent.

Heureusement, les développeurs chez **TMS** ont fait le travail pour nous ! Se basant sur **l'excellent travail** qu'ils ont réalisé sur les Dialogues de Tâche de Vista, ils ont publié un composant nommé **TTaskDialog**. Je ne veux pas faire de cet article une publicité cachée, mais étant donné que notre article original s'intéressait particulièrement aux Dialogues de Tâche (parmi les autres éléments de l'interface utilisateur), j'ai pensé que ce composant méritait d'être mentionné.

Le composant **TTaskDialog** reprend le travail que TMS a déjà réalisé en exposant toutes les fonctionnalités avancées que l'on retrouve dans l'API des Dialogues de Tâche, et apporte un support natif pour les précédent OS comme XP et plus récents. Cela signifie que les développeurs peuvent profiter des avantages de Dialogues de Tâche dès aujourd'hui et continuer à supporter les versions plus récentes de Windows.

X - Conclusion et problèmes en suspens

Tous ces changements rassemblés nous donnent une application dont le support de Windows Vista est amélioré par rapport à l'article original. Nos polices se comporteront maintenant comme elles le doivent lorsque Segoe UI n'est pas disponible, et peuvent éventuellement être modifiées pour s'adapter à la police des icônes du bureau Windows. Nos zones de contenu supportent maintenant un style de police plus récent, et le style de nos contrôles TreeView reprend celui des nouveaux éléments de l'interface utilisateur de Vista.



Application d'exemple mise à jour



[Téléchargez le code d'exemple mis à jour \(miroir\)](#)

[Téléchargez l'application finale \(miroir\)](#)

Cependant, il reste un problème en suspens, et tout retour ou potentielle solution pour ce problème est le bienvenu (voir les contacts en début de page). Si vous combinez **le code du premier article** pour supporter les animations de réduction et de maximisation avec **le code de cet article** pour supporter les nouveaux éléments du TreeView,

votre application possèdera un bug, léger mais détectable. Réduire une fenêtre aura pour effet de d'abord disparaître derrière tout fenêtre immédiatement en-dessous d'elle avant de procéder à l'animation vers la barre de tâches. Ce bug peut également être remarqué dans certaines applications Windows, il est donc difficile de dire ce qui pourrait en être la cause. Tout retour est le bienvenu, et l'auteur d'une solution sera cité.

Ci-dessous, vous trouverez une liste complétée de liens vers des articles et des tutoriels impliquant Windows Vista et interfaces utilisateur. Merci encore à tous pour vos retours et votre accueil fait à l'article original. J'espère que cette mise à jour sera d'une plus value intéressante.

-  [What's New in Windows Vista](#)
-  [Top Rules for the Windows Vista User Experience](#)
-  [Top Guidelines Violations](#)
-  [Top 10 Ways to Light Up Your Windows Vista Apps](#)
-  [The new File Open / Save Dialogs in Windows Vista](#)
-  [Using the new Windows Vista Task Dialog](#)
-  [Taking the New Windows Vista Task Dialog One Step Further](#)
-  [Full Featured Multi-Platform Task Dialog Control](#)

